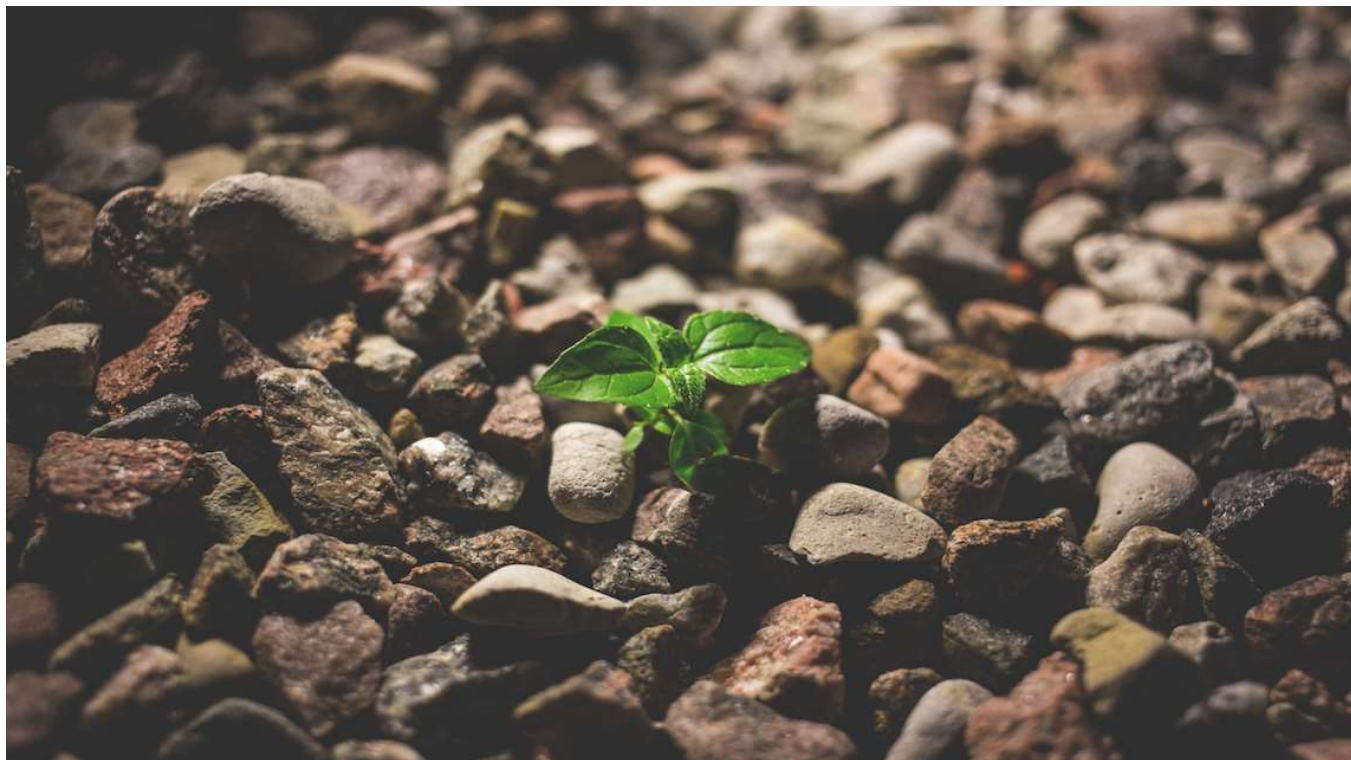


22 | Linux 性能优化答疑（三）

2019-01-09 倪朋飞



22 | Linux 性能优化答疑（三）

朗读人：冯永吉 11'01" | 10.10M

你好，我是倪朋飞。

专栏更新至今，四大基础模块的第二个模块——内存性能篇，我们就已经学完了。很开心你还没有掉队，仍然在积极学习和实践操作，并且热情地留言与讨论。

这些留言中，我非常高兴看到，很多同学用学过的案例思路，解决了实际工作中的性能问题。我也非常感谢 espzest、大甜菜、Smile 等积极思考的同学，指出了文章中某些不当或者不严谨的地方。另外，还有我来也、JohnT3e、白华等同学，积极在留言区讨论学习和实践中的问题，也分享了宝贵的经验，在这里也非常感谢你们。

今天是性能优化的第三期。照例，我从内存模块的留言中摘出了一些典型问题，作为今天的答疑内容，集中回复。为了便于你学习理解，它们并不是严格按照文章顺序排列的。

每个问题，我都附上了留言区提问的截屏。如果你需要回顾内容原文，可以扫描每个问题右下方的二维码查看。

问题 1：内存回收与 OOM

虎虎的这个问题，实际上包括四个子问题，即，

- 怎么理解 LRU 内存回收？
- 回收后的内存又到哪里去了？
- OOM 是按照虚拟内存还是实际内存来打分？
- 怎么估计应用程序的最小内存？

虎虎❤️

写于 2018/12/24

请教老师几个问题

1. 当内存紧张时，系统通过三种机制回收内存。第二种换页比较好理解，但是第一种LRU回收内存页怎么理解？回收后的页去哪了？如果直接删除会导致程序出问题吗？

2. OOM的分数是参照进程的实际消耗内存还是虚拟内存大小？

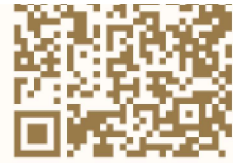
3. 进程启动时，是不是需要分配一个最小的内存？都包括什么呢？如何确定最小的内存是多大呢？比如我在k8s里设置container的request值，我希望能容纳下这个container的最小内存，有没有办法计算呢？

有的问题比较小白，望老师包含。如果无法简短的回答，能否推荐些资料呢？谢谢

引自：Linux性能优化实战

15 | 基础篇：Linux内存是怎么工作的？





其实在 Linux [内存的原理篇](#)和 [Swap 原理篇](#)中我曾经讲到，一旦发现内存紧张，系统会通过三种方式回收内存。我们来复习一下，这三种方式分别是：

- 基于 LRU (Least Recently Used) 算法，回收缓存；
- 基于 Swap 机制，回收不常访问的匿名页；
- 基于 OOM (Out of Memory) 机制，杀掉占用大量内存的进程。

前两种方式，缓存回收和 Swap 回收，实际上都是基于 LRU 算法，也就是优先回收不常访问的内存。LRU 回收算法，实际上维护着 active 和 inactive 两个双向链表，其中：

- active 记录活跃的内存页；
- inactive 记录非活跃的内存页。

越接近链表尾部，就表示内存页越不常访问。这样，在回收内存时，系统就可以根据活跃程度，优先回收不活跃的内存。

活跃和非活跃的内存页，按照类型的不同，又分别分为文件页和匿名页，对应着缓存回收和 Swap 回收。

当然，你可以从 /proc/meminfo 中，查询它们的大小，比如：

```
1 # grep 表示只保留包含 active 的指标（忽略大小写）
2 # sort 表示按照字母顺序排序
3 $ cat /proc/meminfo | grep -i active | sort
4 Active(anon):      167976 kB
5 Active(file):      971488 kB
6 Active:            1139464 kB
7 Inactive(anon):     720 kB
8 Inactive(file):    2109536 kB
9 Inactive:          2110256 kB
```


复制代码

第三种方式，OOM 机制按照 oom_score 给进程排序。oom_score 越大，进程就越容易被系统杀死。

当系统发现内存不足以分配新的内存请求时，就会尝试[直接内存回收](#)。这种情况下，如果回收完文件页和匿名页后，内存够用了，当然皆大欢喜，把回收回来的内存分配给进程就可以了。但如果内存还是不足，OOM 就要登场了。

OOM 发生时，你可以在 dmesg 中看到 Out of memory 的信息，从而知道是哪些进程被 OOM 杀死了。比如，你可以执行下面的命令，查询 OOM 日志：

```
1 $ dmesg | grep -i "Out of memory"
2 Out of memory: Kill process 9329 (java) score 321 or sacrifice child
```

 复制代码

当然了，如果你不希望应用程序被 OOM 杀死，可以调整进程的 oom_score_adj，减小 OOM 分值，进而降低被杀死的概率。或者，你还可以开启内存的 overcommit，允许进程申请超过物理内存的虚拟内存（这儿实际上假设的是，进程不会用光申请到的虚拟内存）。

这三种方式，我们就复习完了。接下来，我们回到开始的四个问题，相信你自己已经有了答案。

1. LRU 算法的原理刚才已经提到了，这里不再重复。
2. 内存回收后，会被重新放到未使用内存中。这样，新的进程就可以请求、使用它们。
3. OOM 触发的时机基于虚拟内存。换句话说，进程在申请内存时，如果申请的虚拟内存加上服务器实际已用的内存之和，比总的物理内存还大，就会触发 OOM。
4. 要确定一个进程或者容器的最小内存，最简单的方法就是让它运行起来，再通过 ps 或者 smap，查看它的内存使用情况。不过要注意，进程刚启动时，可能还没开始处理实际业务，一旦开始处理实际业务，就会占用更多内存。所以，要记得给内存留一定的余量。

问题 2：文件系统与磁盘的区别

文件系统和磁盘的原理，我将在下一个模块中讲解，它们跟内存的关系也十分密切。不过，在学习 Buffer 和 Cache 的原理时，我曾提到，Buffer 用于磁盘，而 Cache 用于文件。因此，有不少同学困惑了，比如 JJ 留言中的这两个问题。

- 读写文件最终也是读写磁盘，到底要怎么区分，是读写文件还是读写磁盘呢？
- 读写磁盘难道可以不经文件系统吗？

JJ

写于 2018/12/26

还是有点困惑，感觉读写磁盘上的数据不就是读写磁盘上的文件里的数据嘛，难道读磁盘上的数据可以不经过文件系统吗，可以直接读裸磁盘？有点没理解buffer是磁盘上的数据缓存，cache是文件数据缓存，求大神解答下。。

引自：Linux性能优化实战

16 | 基础篇：怎么理解内存中的Buffer和Cache？

识别二维码打开原文
「极客时间」App



如果你也有相同的疑问，主要还是没搞清楚，磁盘和文件的区别。我在“[怎么理解内存中的Buffer 和 Cache](#)”文章的留言区简单回复过，不过担心有同学没有看到，所以在这里重新讲一下。

磁盘是一个存储设备（确切地说是块设备），可以被划分为不同的磁盘分区。而在磁盘或者磁盘分区上，还可以再创建文件系统，并挂载到系统的某个目录中。这样，系统就可以通过这个挂载目录，来读写文件。

换句话说，磁盘是存储数据的块设备，也是文件系统的载体。所以，文件系统确实还是要通过磁盘，来保证数据的持久化存储。

你在很多地方都会看到这句话，Linux 中一切皆文件。换句话说，你可以通过相同的文件接口，来访问磁盘和文件（比如 open、read、write、close 等）。

- 我们通常说的“文件”，其实是指普通文件。
- 而磁盘或者分区，则是指块设备文件。

你可以执行 “ls -l < 路径 >” 查看它们的区别。如果不懂 ls 输出的含义，别忘了 man 一下就可以。执行 man ls 命令，以及 info ‘(coreutils) ls invocation’ 命令，就可以查到了。

在读写普通文件时，I/O 请求会首先经过文件系统，然后由文件系统负责，来与磁盘进行交互。而在读写块设备文件时，会跳过文件系统，直接与磁盘交互，也就是所谓的“裸 I/O”。

这两种读写方式使用的缓存自然不同。文件系统管理的缓存，其实就是 Cache 的一部分。而裸磁盘的缓存，用的正是 Buffer。

更多关于文件系统、磁盘以及 I/O 的原理，你先不要着急，往后我们都会讲到。

问题 3：如何统计所有进程的物理内存使用量

这其实是 [怎么理解内存中的 Buffer 和 Cache](#) 的课后思考题，无名老卒、Griffin、JohnT3e 等少数几个同学，都给出了一些思路。

比如，无名老卒同学的方法，是把所有进程的 RSS 全部累加：

无名老卒

写于 2018/12/27

看了这篇文章，终于理解 了buffers以及cache，之前在网上还专门查过这2者的区别，但就是像老师说的那样，文章看下来，啥也没有啥明白。

按照老师的总结，cache是针对文件系统的缓存，而buffers是对磁盘数据的缓存，是直接跟硬件那一层相关的，那一般来说，cache会比buffers的数量大了很多。生产环境下面看了多台机器，的确如此。

后面留的那个作业，如果要统计一个进程所占用的物理空间，我的做法是累加RSS的值。如下shell是我工作中所使用的命令，取内存占用top10的进程：

```
for i in $( ls /proc/ |grep "[0-9]" |awk '$0 >100' ) ;
do cmd="";[ -f /proc/$i/cmdline ] && cmd=`cat /p
roc/$i/cmdline`;[ "$cmd"X = ""X ] && cmd=$i;aw
k -v i="$cmd" '/Rss:/{a=a+$2}END{printf("%s:%
d\n",i,a)}' /proc/$i/smmaps 2>/dev/null; done | sort
-t: -k2nr | head -10
```


识别二维码打开原文
「极客时间」App



这种方法，实际上导致不少地方会被重复计算。RSS 表示常驻内存，把进程用到的共享内存也算了进去。所以，直接累加会导致共享内存被重复计算，不能得到准确的答案。

留言中好几个同学的答案都有类似问题。你可以重新检查一下自己的方法，弄清楚每个指标的定义和原理，防止重复计算。

当然，也有同学的思路非常正确，比如 JohnT3e 提到的，这个问题的关键在于理解 PSS 的含义。

JohnT3e

写于 2018/12/26

最后的思考题，关键点在于理解PSS项的含义。
有兴趣的同学可以参考这个资料：
<https://unix.stackexchange.com/questions/33381/getting-information-about-a-process-memory-usage-from-proc-pid-smaps>。

引自：Linux性能优化实战

16 | 基础篇：怎么理解内存中的Buffer和Cache？

识别二维码打开原文
「极客时间」App



你当然可以通过 stackexchange 上的[链接](#)找到答案，不过，我还是更推荐，直接查 proc 文件系统的[文档](#)：

The “proportional set size” (PSS) of a process is the count of pages it has in memory, where each page is divided by the number of processes sharing it. So if a process has 1000 pages all to itself, and 1000 shared with one other process, its PSS will be 1500.

这里我简单解释一下，每个进程的 PSS，是指把共享内存平分到各个进程后，再加上进程本身的非共享内存大小的和。

就像文档中的这个例子，一个进程的非共享内存为 1000 页，它和另一个进程的共享进程也是 1000 页，那么它的 $PSS = 1000/2 + 1000 = 1500$ 页。

这样，你就可以直接累加 PSS，不用担心共享内存重复计算的问题了。

比如，你可以运行下面的命令来计算：

```
1 # 使用 grep 查找 Pss 指标后，再用 awk 计算累加值
2 $ grep Pss /proc/[1-9]*/smaps | awk '{total+=$2}; END {printf "%d kB\n", total }'
3 391266 kB
```

 复制代码

问题 4: CentOS 系统中如何安装 bcc-tools

很多同学留言说用的是 CentOS 系统。虽然我在文章中也给出了一个[参考文档](#)，不过 bcc-tools 工具安装起来还是有些困难。

比如白华同学留言表示，网络上的教程不太完整，步骤有些乱：

白华

写于 2018/12/28

老师，centos7版本安装这个软件能不能出篇文章，看github上的安装教程有些乱，百度看文章也很少，需要下载的内容有很多都无法下载，根本无法进行试验操作

引自：Linux性能优化实战

171 案例篇：如何利用系统缓存优化程序的运行效率？

识别二维码打开原文
「极客时间」App



不过，白华和渡渡鸟_linux 同学在探索实践后，留言分享了他们的经验，感谢你们的分享。

白华

写于 2018/12/29

centos7系统安装bcc-tools的教程我写在了简书上：<https://www.jianshu.com/p/997e0a6d8e09>
大家如果有安装不下来的可以看看

引自：Linux性能优化实战

171 案例篇：如何利用系统缓存优化程序的运行效率？

识别二维码打开原文
「极客时间」 App



渡渡鸟_linux

补充下centos7使用yum 安装bcc-tools:

```
[root@centos-80 ~]# yum update
```

```
[root@centos-80 ~]# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org && rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm
```

```
[root@centos-80 ~]# uname -r ##  
3.10.0-862.el7.x86_64
```

```
[root@centos-80 ~]# yum remove kernel-headers kernel-tools kernel-tools-libs
```

```
[root@centos-80 ~]# yum --disablerepo="*" --enablerepo="elrepo-kernel" install kernel-ml kernel-ml-devel kernel-ml-headers kernel-ml-tools kernel-ml-tools-libs kernel-ml-tools-libs-devel
```

```
[root@centos-80 ~]# sed -i '/GRUB_DEFAULT/s/=.*/=0/' /etc/default/grub
```

```
[root@centos-80 ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

```
[root@centos-80 ~]# reboot
```

```
[root@centos-80 ~]# uname -r ## 升级成功  
4.20.0-1.el7.elrepo.x86_64
```

```
[root@centos-80 ~]# yum install -y bcc-tools
```

```
[root@centos-80 ~]# echo 'export PATH=$PATH:/usr/share/bcc/tools' > /etc/profile.d/bcc-tools.sh
```

```
[root@centos-80 ~]# . /etc/profile.d/bcc-tools.sh
```



```
[root@centos-80 ~]# cachestat 1 1 ## 测试安装是否成功
```

| TOTAL RS_MB | MISSES CACHED_MB | HITS | DIRTIES | BUFFE |
|----------------|---------------------|------|---------|-------|
|----------------|---------------------|------|---------|-------|

| | | | | |
|---|---|---|---|-------|
| 0 | 0 | 0 | 0 | 2 287 |
|---|---|---|---|-------|

引自：Linux性能优化实战

171 案例篇：如何利用系统缓存优化程序的运行效率？

识别二维码打开原文
「极客时间」App



在这里，我也统一回复一下，在 CentOS 中安装 bcc-tools 的步骤。以 CentOS 7 为例，整个安装主要可以分两步。

第一步，升级内核。你可以运行下面的命令来操作：


```
1 # 升级系统
2 yum update -y
3
4 # 安装 ELRepo
5 rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
6 rpm -Uvh https://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
7
8 # 安装新内核
9 yum remove -y kernel-headers kernel-tools kernel-tools-libs
10 yum --enablerepo="elrepo-kernel" install -y kernel-ml kernel-ml-devel kernel-ml-headers kernel-ml
11
12 # 更新 Grub 后重启
13 grub2-mkconfig -o /boot/grub2/grub.cfg
14 grub2-set-default 0
15 reboot
16
```

复制代码

```
17 # 重启后确认内核版本已升级为 4.20.0-1.el7.elrepo.x86_64
18 uname -r
```

第二步，安装 bcc-tools：

```
1 # 安装 bcc-tools
2 yum install -y bcc-tools
3
4 # 配置 PATH 路径
5 export PATH=$PATH:/usr/share/bcc/tools
6
7 # 验证安装成功
8 cachestat
```

 复制代码

问题 5：内存泄漏案例的优化方法

这是我在 [内存泄漏了，我该如何定位和处理](#) 中留的一个思考题。这个问题是这样的：

在内存泄漏案例的最后，我们通过增加 `free()` 调用，释放了函数 `fibonacci()` 分配的内存，修复了内存泄漏的问题。就这个案例而言，还有没有其他更好的修复方法呢？

很多同学留言写下了自己的想法，都很不错。这里，我重点表扬下郭江伟同学，他给出的方法非常好：

郭江伟

本例中将动态分配内存改为使用数组，然后就不需要自己free了；

将app.c拷贝为app2.c 做如下修改，因为篇幅有限没法贴完全代码：

```
long long fibonacci(long long *n0, long long *n1)
{
```

```
    //分配1024个长整数空间方便观测内存的变化情况
```

```
    //    long long *v = (long long *) calloc(1024, sizeof(long long));
```

```
    long long v[1024];
```

然后执行memleak

```
gjl@gjl:~$ sudo /usr/share/bcc/tools/memleak -p $(pidof app2c)
```

```
Attaching to pid 3463, Ctrl+C to quit.
```

```
[13:02:24] Top 10 stacks with outstanding allocations:
```

```
[13:02:29] Top 10 stacks with outstanding allocations:
```

```
^Cgjl@gjl:~$ sudo /usr/share/bcc/tools/memleak -p $(pidof app2c)
```

```
Attaching to pid 3463, Ctrl+C to quit.
```

```
[13:02:43] Top 10 stacks with outstanding allocations:
```

```
[13:02:48] Top 10 stacks with outstanding allocations:
```

[13:02:53] Top 10 stacks with outstanding allocations:

[13:02:58] Top 10 stacks with outstanding allocations:

引自：Linux性能优化实战

181 案例篇：内存泄漏了，我该如何定位和处理？

识别二维码打开原文
「极客时间」App



他的思路是不用动态内存分配的方法，而是用数组来暂存计算结果。这样就可以由系统自动管理这些栈内存，也不存在内存泄漏的问题了。

这种减少动态内存分配的思路，除了可以解决内存泄漏问题，其实也是常用的内存优化方法。比如，在需要大量内存的场景中，你就可以考虑用栈内存、内存池、HugePage 等方法，来优化内存的分配和管理。

除了这五个问题，还有一点我也想说一下。很多同学在说工具的版本问题，的确，生产环境中的Linux 版本往往都比较低，导致很多新工具不能在生产环境中直接使用。

不过，这并不代表我们就无能为力了。毕竟，系统的原理都是大同小异的。这其实也是我一直强调的观点。

- 在学习时，最好先用最新的系统和工具，它们可以为你提供更简单直观的结果，帮你更好的理解系统的原理。
- 在你掌握了这些原理后，回过头来，再去理解旧版本系统中的工具和原理，你会发现，即便旧版本中的很多工具并不是那么好用，但是原理和指标是类似的，你依然可以轻松掌握它们的使用方法。

最后，欢迎继续在留言区写下你的疑问，我会持续不断地解答。我的目的不变，希望可以和你一起，把文章的知识变成你的能力，我们不仅仅在实战中演练，也要在交流中进步。



Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师
Kubernetes 项目维护者



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©版权归极客邦科技所有，未经许可不得转载

上一篇 21 | 套路篇：如何“快准狠”找到系统内存的问题？

写留言

精选留言



白华

👍 1

自己跑虚拟机跑的k8s小集群，node节点跑的镜像太多，就特别卡，看集群情况发现好多pod都死了，看虚拟机上面就写到oom自动杀进程了，以前从没遇到过oom，这次一下就知道怎么回事了

2019-01-09



风飘，吾独思
打卡

👍 0

2019-01-09



ninuxer

👍 0

打卡day23

喜欢一篇文章看两次，第一次看，第二次是实践的时候再看一遍

2019-01-09